Challenges for Compiler Research

The Implications of Increasing Complexity of Platforms, Languages, and Applications

Ken Kennedy

Rice University

http://www.cs.rice.edu/~ken/Presentations/CompilerChallenges.pdf

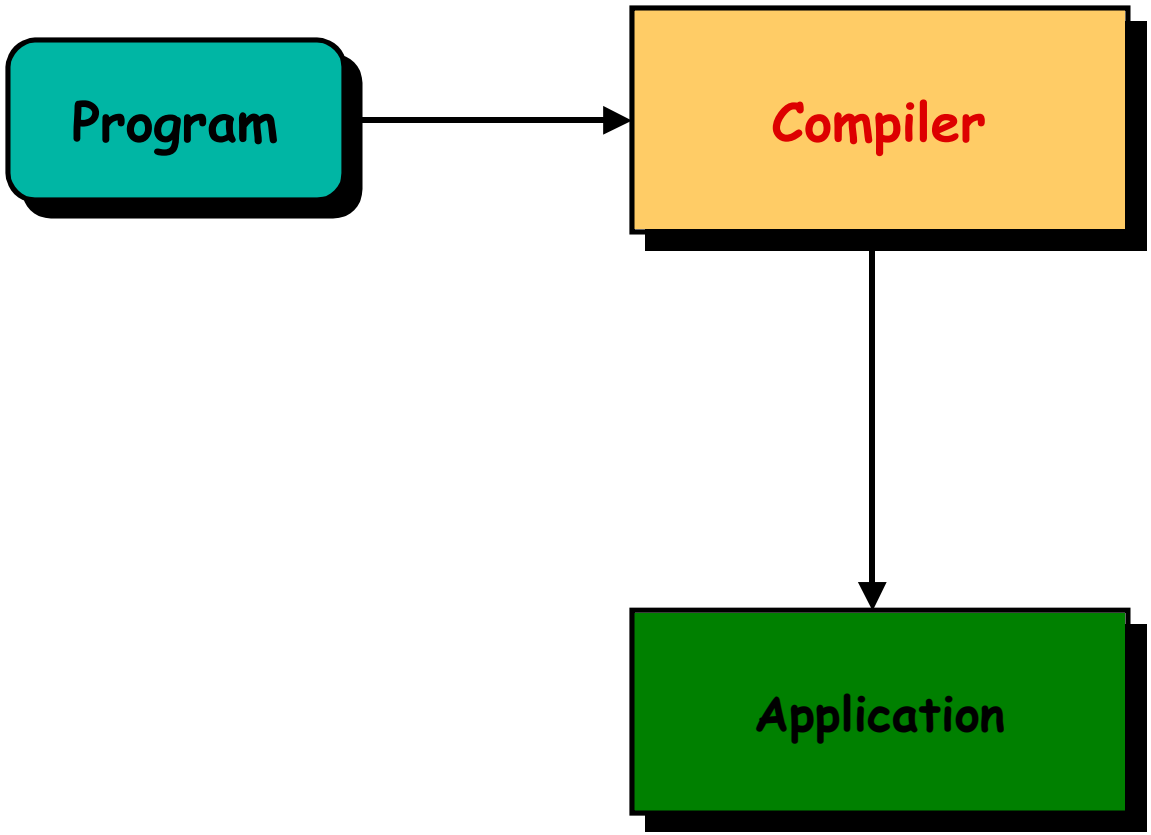
Driving Forces

- Platform Complexity
  - Parallelism
  - Memory hierarchy
  - Grids
- Application Complexity
  - Sheer size
  - Diversity of components, languages, paradigms
  - Application composition
    - MADIC study: 10,000 apps, untrusting developers
  - Need for software reliability
- Shortage of Professional Developers
  - Need for greater productivity

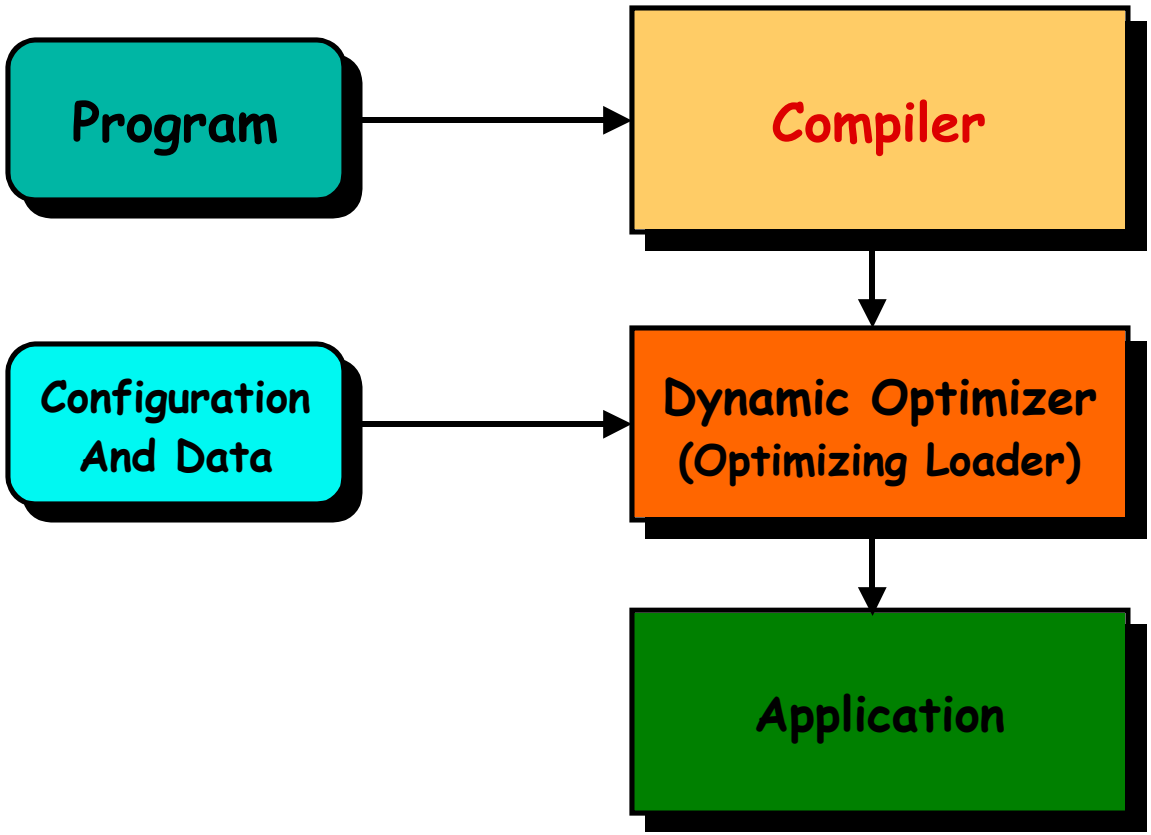
Implications I

- Many traditional compiler decisions will be made late
  - Grid: Target platforms known just before execution
    - May need to change during execution
    - Applications will need to adapt (but should the user do this)
  - Optimizations may be determined at run time
    - In response to problem data
- Compilation is becoming a process consisting of many steps
  - Steps take place at different times in the program preparation process
    - When procedure compiled
    - When program defined
    - When platform(s) determined
    - When data known

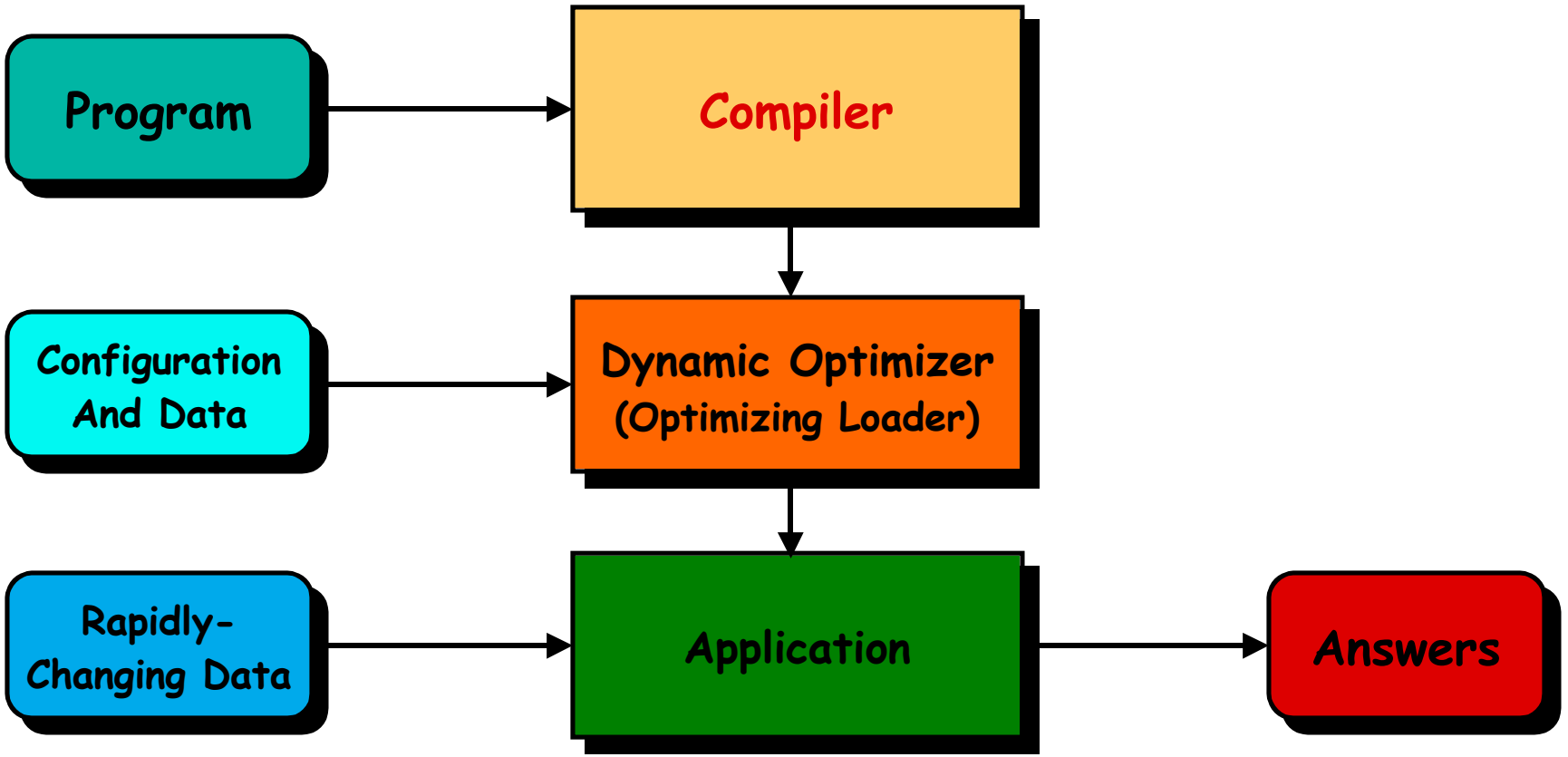
Dynamic Optimization



Dynamic Optimization

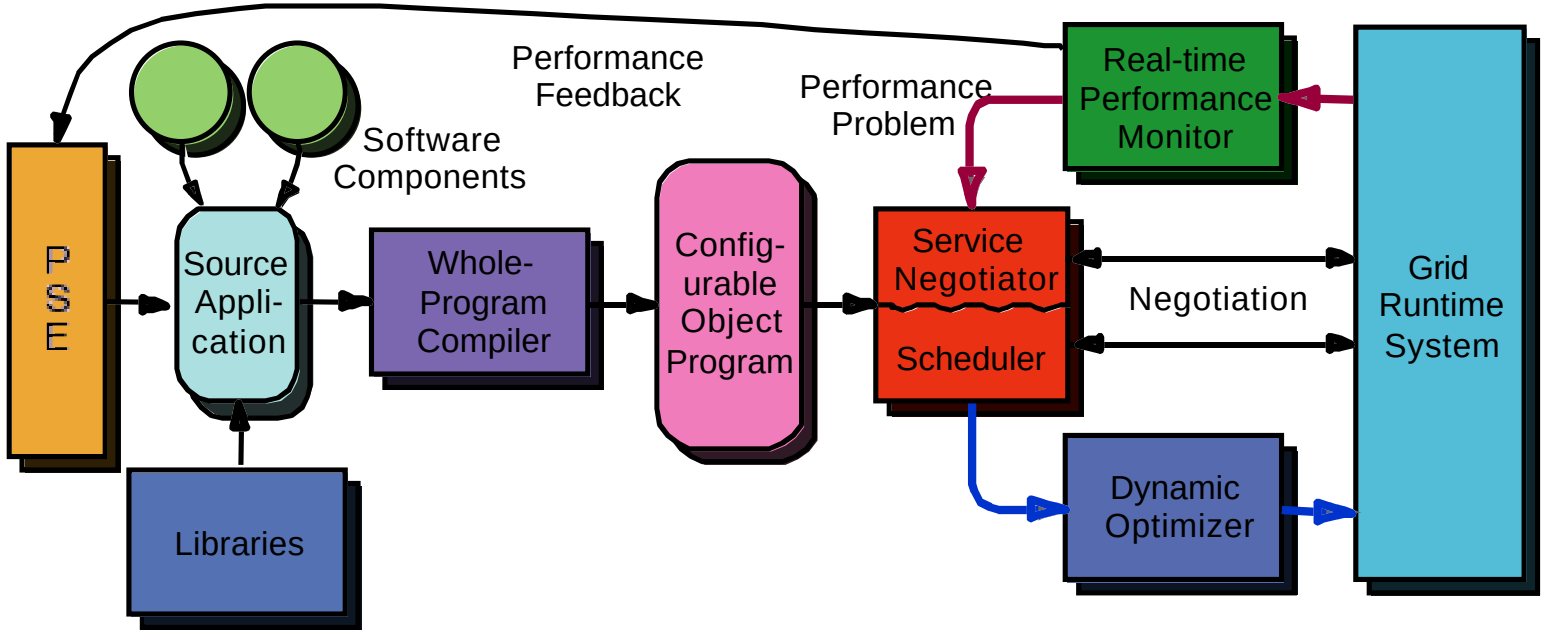


Dynamic Optimization



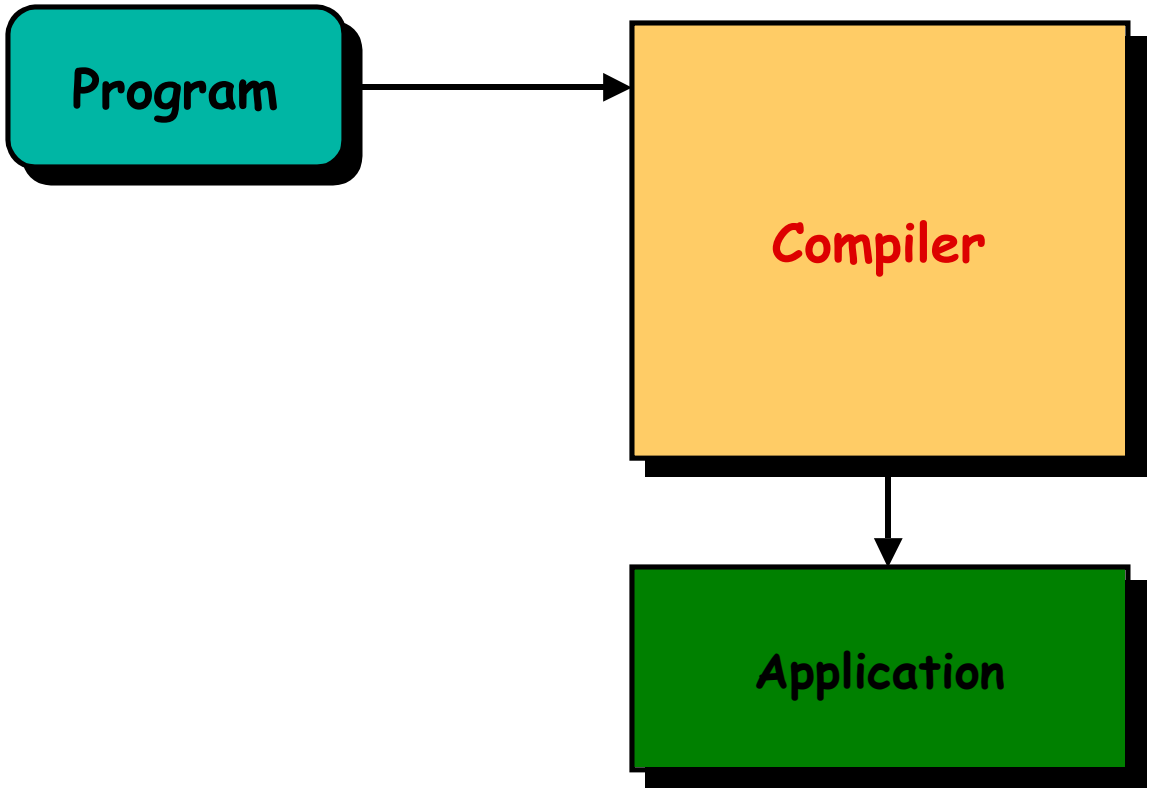
Grid Compilation Architecture

- **Goal:** reliable performance under varying load

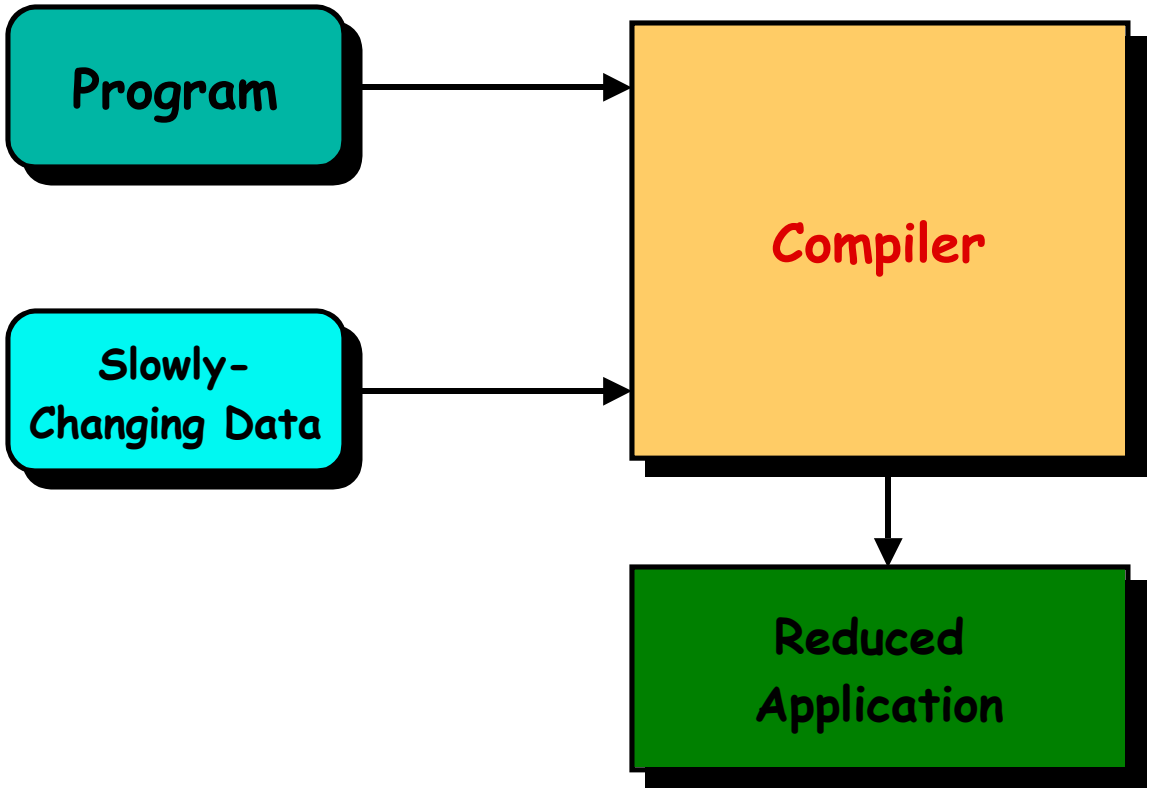


GrADS Project: Berman, Chien, Cooper, Dongarra, Foster, Gannon, Johnsson, Kennedy, Kesselman, Mellor-Crummey, Reed, Torczon, Wolski

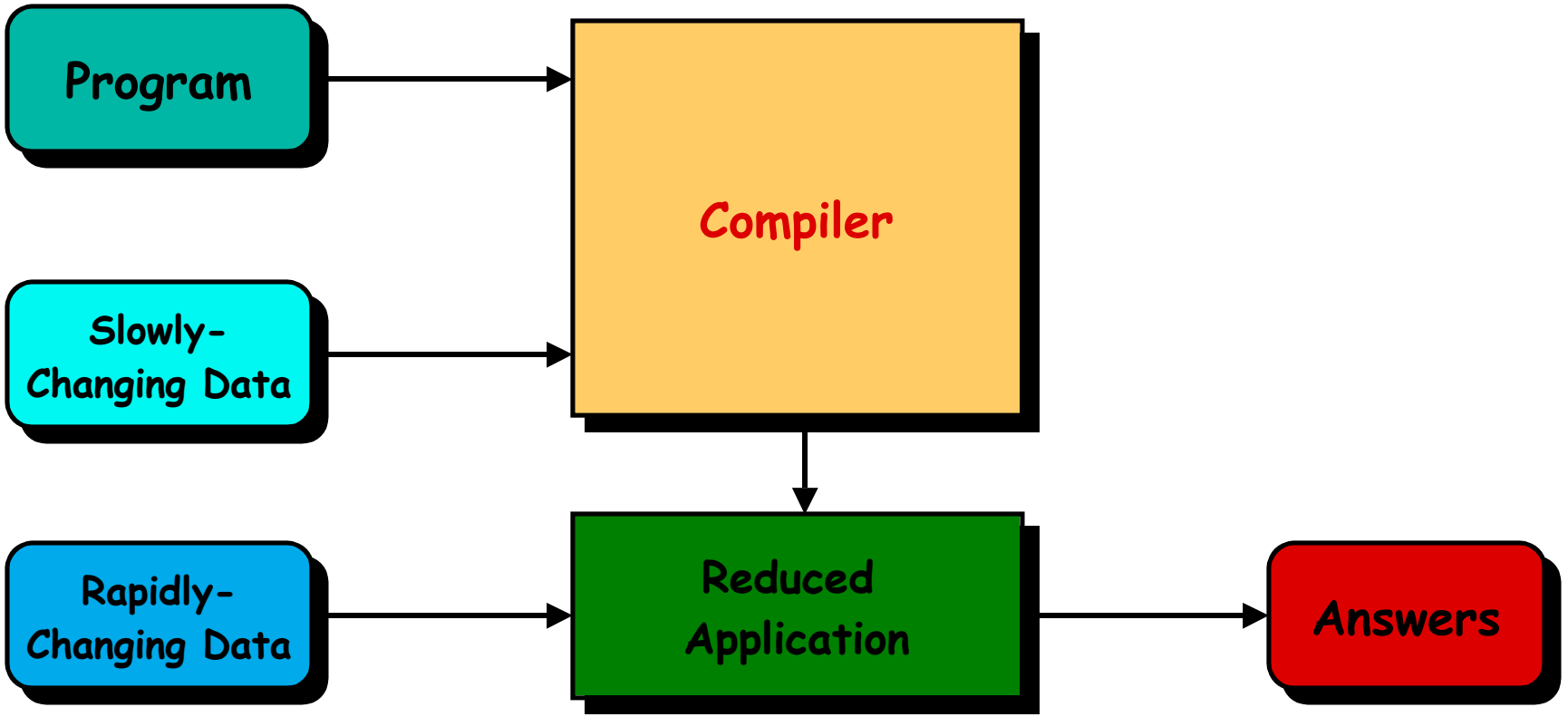
Compiling with Data



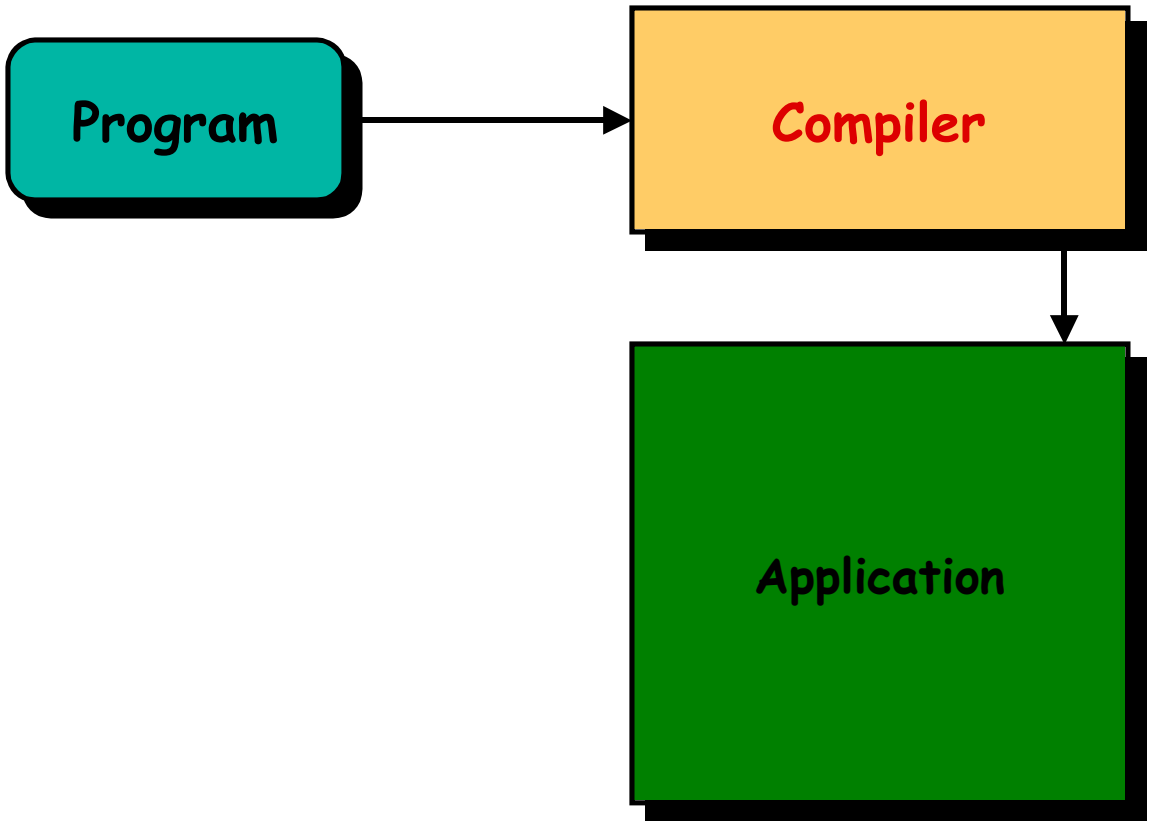
Compiling with Data



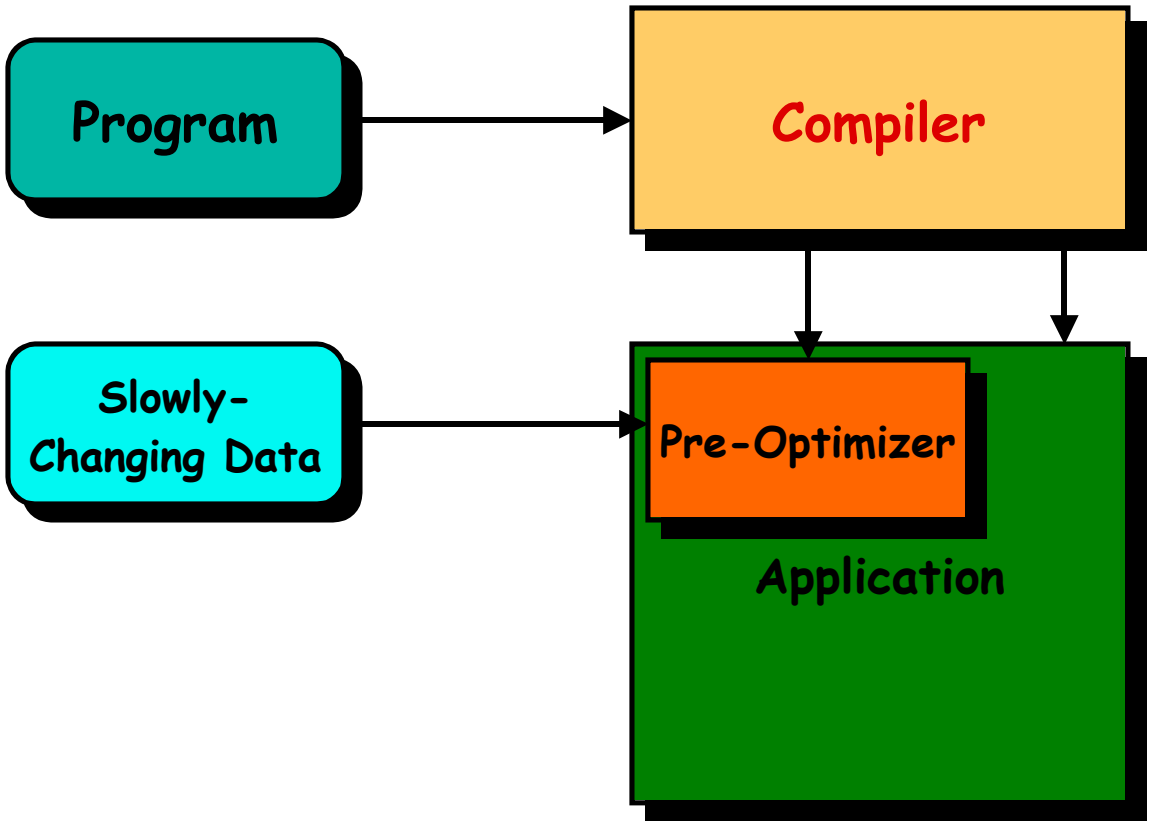
Compiling with Data



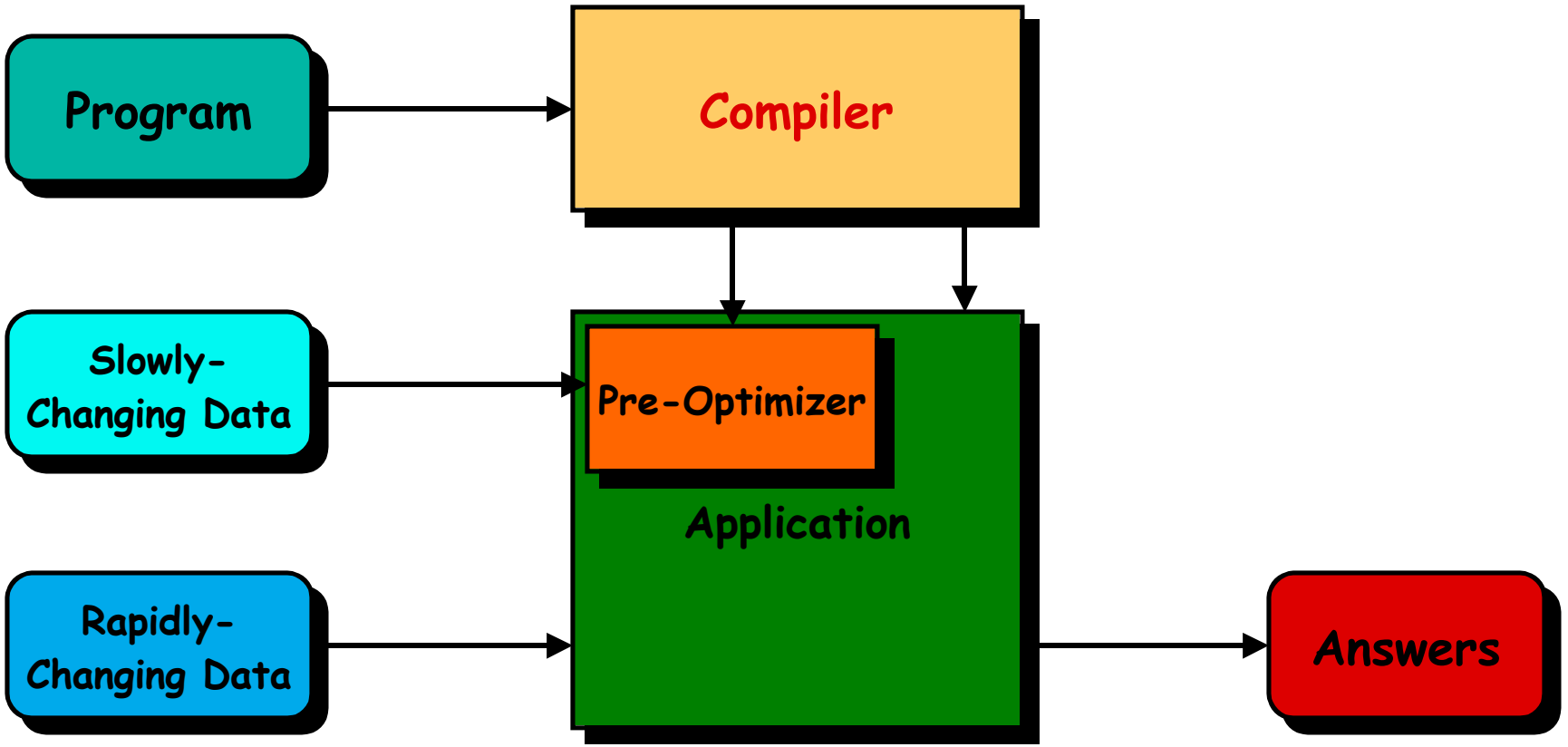
Run-Time Compilation



Run-Time Compilation



Run-Time Compilation

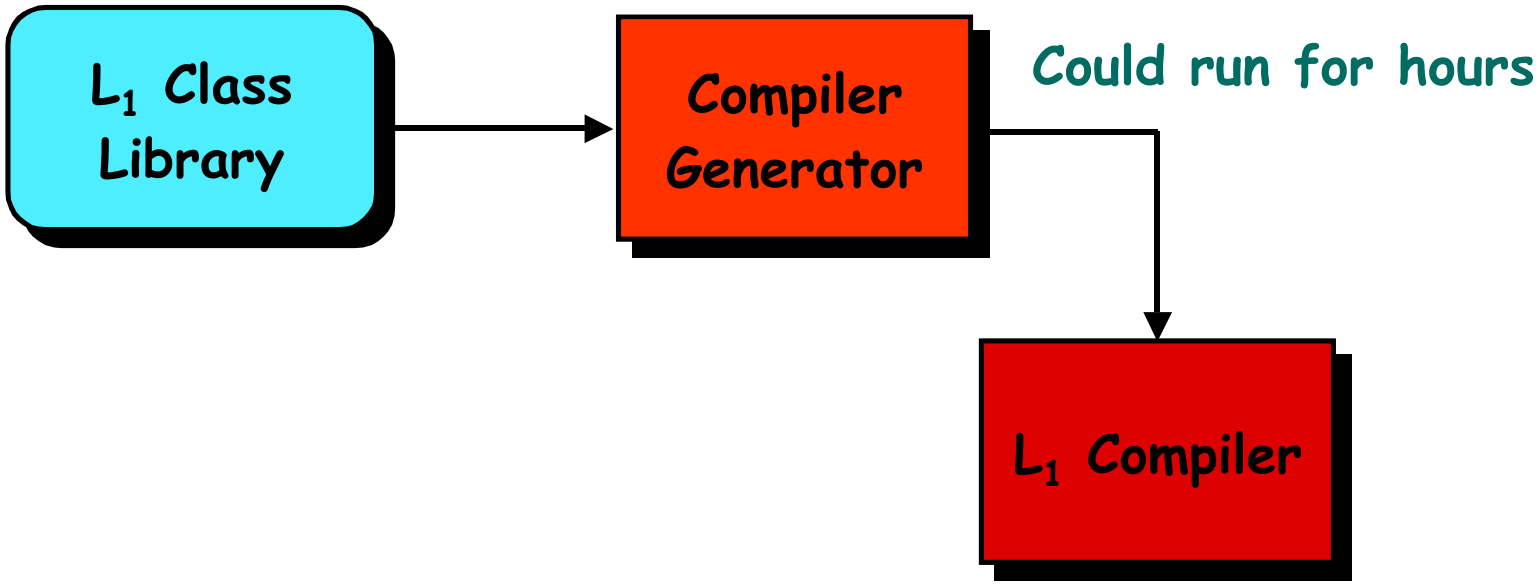


Implications II

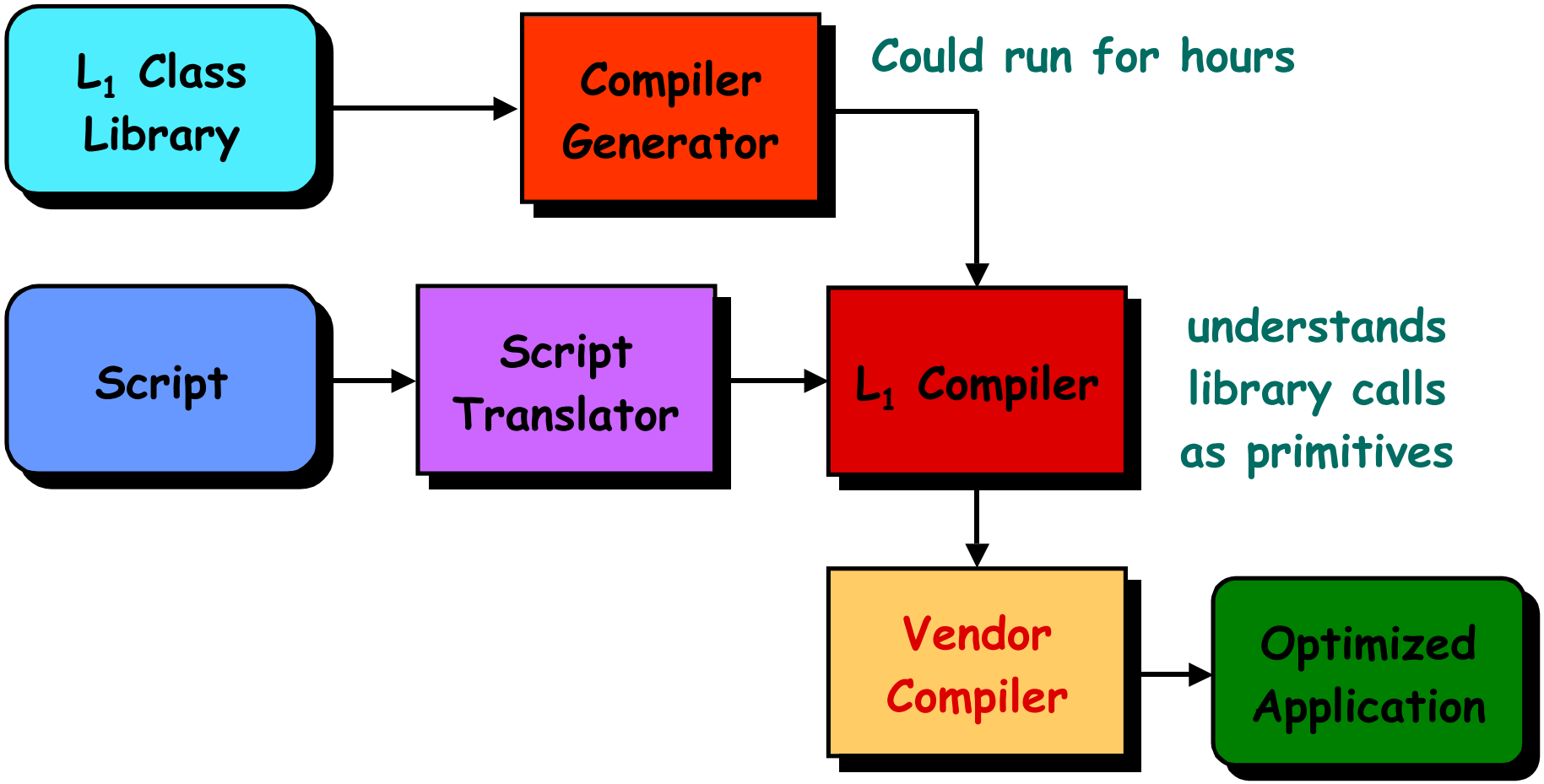
- Need to provide support for application migration across platforms
  - Self-tuning applications
    - Examples: Atlas, FFTW, …
  - Throw computer time at the problem
- Need to involve the end user in application development
  - More focus on high-level domain-specific programming systems
    - Example: popularity of Matlab
  - Strategy: High-level programming systems based on libraries coded by experts
  - Problem of performance
    - Optimizations should treat libraries as language primitives

Telescoping Languages

Telescoping Languages



Telescoping Languages



Implications III

- Compiler technology can play a role in reliability
  - Reliability and security of networked systems must be a national priority
  - Many security and reliability problems arise from hand optimizations in context
    - Example: stack overflow in server software
  - Compilers can perform many of these optimizations mechanically on carefully coded modules intended for general use
    - If we can make compiler technology reliable

Summary

- Target platforms and applications continue to become more complicated
  - Complexity for compiler
  - Complexity for user
- Shortage of programming talent
  - Throw computer time at the problem of platform migration
  - Involve end users in programming
  - Use experts for components, compilers for optimization
- Reliability and security must be increased
  - Mechanical optimization in context can help
    - But only if compilers can be made reliable